The Acorn Atom

by Jim Murray

THE QUESTION uppermost in my mind when I first received Atom for review was: "At whom has Acorn aimed the machine"? I took delivery of the machine in a cardboard casing measuring 17in. by 12in. by 4in. which housed the unit supported by foam mountings plus the three necessary electrical cables and the single accompanying manual.

The computer is of the single-unit type — a formed plastic shell supporting the processor, memory, interfaces and a full-sized QWERTY keyboard. It is also small enough to pack into an average-sized briefcase. Having no built-in display or printer, the Atom uses what surely must be the best method of output for low-cost systems — direct connection to the aerial socket of a domestic TV set.

After unpacking the unit, and turning to the manual, I met the first snag; there was no indication anywhere as to what voltage should be applied to the power line. Fortunately, there was a note written on the transit box: "Needs 5V — two 2.5V batteries". So I duly connected it to my own clean 5V supply. In fairness to the manufacturer, I learned subsequently that the review machine was a pre-production model although the manual supplied was, in fact, the latest edition.

Hardware components

Establishing that with the correct power applied, the unit gave the proper prompt when connected to the TV, I turned my attention to the hardware components. The available configurations range from the minimum system which is an 8Kbyte monitor plus 12K byte RAM. The review system had the 12Kbyte monitor and 12Kbyte RAM.

The expanded system also supports a floating-point option which was included in the review machine. Without that option, the user is limited to using integer-only arithmetic. Having verified that the machine functioned correctly, it was time to look inside and examine the architecture. The nucleus of the Atom is the 6502 processor chip, now established as one of the standards, at the heart of the Apple, the Pet and the Rockwell AIM systems. For the assembler programmer, the 6502 instruction set provides some useful facilities especially in its addressing modes, and the idiomatic Basic provided with the Atom denies the Basic programmer very few of those facilities. Memory was provided using 24 2114 4K static RAMs.

Input/output other than through the built-in keyboard, TV VDU and cassette channels was through a 6522 VIA — versatile interface adaptor — which provides two parallel ports, interval timers and a serial/parallel converter. Finally, the display processor was a 6847 which allowed a range of display options, all user-selectable using the standard software provided.

Eight graphics modes are available in all but only the base level mode — mode 0 — is available on the unexpanded machine. In that mode, the screen is structured as 16 lines of 32 columns — each character is either one of the standard 64 ASCII set or one of the standard 64 special symbols created from a six-element Pixel.

The ASCII characters can be displayed either in the standard way — white on black — or in reverse video, and the special symbols can be shown as black and white or black and grey. Mode 0 allows the full range of allowed characters and symbols to be displayed simultaneously. The higher graphics modes provide progressively higher point addressing on the screen up to a maximum resolution of 256 by 192 in black and white or 128 by 192 in four colours. However, the higher-resolution and colour facility has to be paid for by the use of up to 6K of the user RAM.

I approached the user manual with some degree of trepidation. For, as many may have experienced, some manuals supplied with hardware and software products are grossly inadequate, others are little short of disgraceful — hastily-produced adjuncts to the product they describe.

Users' manual

No such criticism can be made against the Atom manual, although the title is innaccurate, or at least misleading: "Atomic theory and practice — a beginners' course in Basic and machine-code programming". It implies that, having mastered the contents of the tome, the raw beginner would have acquired a command of both the Basic and 6502 assembler languages.

I feel that to be misleading for two reasons because the version of Basic provided with this machine is highly idiomatic and there are more concepts involved in writing assembler or machine-code programs than are introduced in this manual.

The risk is, therefore, that the beginner might emerge from his thorough study of the Atom monitor as taught by the manual, under the misapprehension that he could program in a standard Basic and even start work with a standard assembler.

With those reservations, it must be said the manual, as an explanation of the Atom monitor, is a masterpiece of clarity — full marks go to David Johnson-Davies for his very readable and instructive work.

There are three sections to the book, dealing with Basic, assembler and general reference points. Each section is neatly divided into chapters, each of which addresses one concept; there are 27 chapters. In sections one and two each chapter then follows the same pattern: a friendly introduction, aimed at the less-experienced reader, explains each new point — loops, arrays, strings, etc. — then coding is defined and finally a set of examples presented.

The worked examples distributed throughout the manual are worthy of a special mention. A wide range is presented and within each chapter, they vary from the simple, which illustrate well the chapter's central theme, to those which present methods using the theme which many career programmers could usefully add to their repertoire of techniques.

A thoughtful overall introduction is provided at the beginning of the manual which suggests a reading for those of different skill levels. The 10-page index at the back of the book is comprehensive.

Important concept

The three software components of the system are the monitor, the Basic interpreter and the assembler. The assembler is accessed through Basic which in turn is accessed via the monitor.

The monitor effectively manages a dynamic area of memory called the text space which grows or shrinks as text is added to or deleted from it. Any user input line preceded by a line number goes into the text space either as a new line or to replace an existing line with the same number. The text lines are sorted on line number as generally the text space will hold the currently-active program — either Basic or assembler.

An important concept is multiple text spaces. At any point, the user can redefine the start of the text space he wishes to manipulate and can thus have resident in memory several areas all containing lines with the same range of line numbers. Those text spaces may perhaps hold programs or there may be a mixture of programs and data. Regardless of what they hold, however, only one can be active and accessible by the monitor at any time.

The monitor thus acts as the text space — program — editor, but the only facilities offered are whole-line replacement and selective lists. There are no utilities which provide string searches, string replacement of re-sequencing.

The manual provides, however, an example of a program in a secondary text space which re-numbers the main program, resident in the primary text space an the enterprising programmer may wish to equip himself with similar utilities to emulate his favourite editor before embarking on a major project.

Text spaces are saved and loaded to and from cassette tape via simple monitor commands which allow the user to assign file names to the tape files, the encoding format being CUTS — or Kansas City Standard — with a reasonably modest fixed speed of 300 baud.

I tried three different cassette recorders to test the cassette system. They varied in price from £25 to £55. Only on the most expensive recorder did I achieve consistently trouble-free data transfer.

While it might be argued that a good quality recorder is a good investment for those wishing to use it for off-line data storage, I feel one must consider the position of the purchaser of a low-cost system such as the Acorn Atom. To him, the cost of upgrading his tape recorder must represent a large proportion of the total cost of the system.

I believe, however, that the explanation of this particular fault lay once again in the fact that I was using a pre-production system — I saw another later Atom connected successfully to a low-price recorder.

The general point is obvious: the buyer of any piece of technical equipment must satisfy himself that it is fully compatible with components he already owns and wishes to use with it.

The Atom text space can contain any data the user wishes to enter. So, when the data is, in fact, a program, no syntax checking can be performed until the run command is issued. At this point the text is interpreted as Basic.

While many, if not most Basics on small machines have eccentricities and can be said to be non-standard, some are positively more non-standard than others. The Atom Basic is definitely non-standard and that, I think, is due to the strong link between it and the built-in assembler.

The user is provided with 26 simple variables, A-Z which when undimensioned can hold either a four-byte integer or four-byte floating-point number if the floating-point option is included. Note that numbers are held in binary, not as ASCII strings, which, of course, makes for higher-speed arithmetic calculation. Thus, the default precision is 32 bits but examples in the manual indicate how to manipulate numbers of arbitary precision if this is needed.

If the simple variables are dimensioned, they are assumed to be byte arrays, holding numbers of eight-bit precision or, more normally, character strings. Those arrays must either be pre-defined or assigned dynamically to unused memory above the program space at program run time. In practice, there is not much difference between the standard dimension statement and dynamic assignment.

Array variables

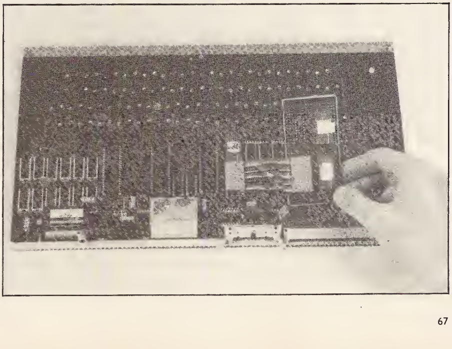
An additional 26 array variables, AA - ZZ, are also provided to hold word arrays. In Atom terms, a word is the four-byte unit. Arrays can have only a single dimension. That appears restrictive until the concept of indirection and word vectors is introduced.

Indirection is a concept more familiar to assembler programmers than to those who normally use Basic but all it means is that a variable contains not a value but a pointer to a value which is held elsewhere. Atom Basic allows word vectors to hold such indirection pointers to other arrays or vectors. Thus, multi-dimensioned arrays can be simply constructed as arrays of arrays.

Admittedly, the penalty is a certain lack of clarity in the program but the bonus is that the inventive programmer can construct rich data structures found usually in the more powerful langauges. Naturally, the Atom Basic provides new operators to deal with these structures.

In the same vein, string variables are handled in a way which pays little allegiance to the more traditional methods, but once again, all the regular string manipul-

(continued on next page)



(continued from previous page)

ation functions are either present or can be very simply constructed. I must admit that initially, I found addressing the various data types rather awkward but after realising the strong association with the processor addressing modes, normally evident only when programming in assembler, all became clear.

The next point of interest was the structure of the executable program statements. Although the screen width is only 32 characters, a Basic line can in fact be 64 characters long. On the screen, the line wraps around after the 32nd character automatically.

Each line can hold as many statements — separated by semicolons — as will fit, and when one statement is an IF test, either all the remaining statements on the line are executed or none, depending on the result of the test. That, of course, helps the programmer as he can dispense with many short conditional GOTO branches whose only purpose is to include or omit a few lines of code. It also enhances the readability of the program.

Reserved words

Another feature, however, while just as valuable in other circumstances detract from the program's readability. That is that most reserved words can be reduced to one or two, sometimes three characters. A multi-statement line using the abbreviations can thus be reasonably complex, and the decision to write clear or concise code is in the hands of the user. A spin-off here is that a powerful immediate mode command can be input and can even involve more than one leve of FOR NEXT loops. Immediate mode commands are a feature of most Basics and normally allow the user to execute any single-line statement which contains no reference to another line without having to declare that single line as a program.

On the subject of loops, the FOR NEXT STEP construction naturally is present, augmented by the more recent DO UNTIL variation and FOR NEXT, DO UNTIL plus GOSUB RETURN sections can be nested to a depth of 15. GOSUBs have two interesting peculiarities. Firstly, the target line — or the target line of a GOTO — can be referenced by a label as well as by line numbers. That, of course, makes the program insensitive to line number changes and, it is claimed, results in faster execution of the program. Indeed, a simple two-line example program which produces a continuous tone on the internal loud speaker proves this, since the variation using a line number as the GOTO target produces a lower note than the variation which uses a line label.

Secondly, GOSUBs can be recursive. For those not totally familiar with recursion, it is simply the concept of a subroutine calling itself. Many classic problems can be very elegantly formulated using this concept and several useful examples are given including the Eight Queens' problem and the Tower of Hanoi, and mercifully excluding the totally useless evaluation of factorial.

Both the Eight Queens and the Tower examples show a continuous graphic display of the solution steps and it is perhaps illuminating to note that both can be run on the minimum 512-byte, user-RAM configuration.

The continuous display brings one to the graphic capabilities of the system which must be a major attraction to potential buyers. The ranges of resolution, or point addressability, mentioned and three verbs, MOVE, PLOT and DRAW, are available which produce point to point movements of the graphics cursor.

If the graphics is to be used, there are two options, one of which illustrates another neat feature of the system. If the extension ROM which processes the fourth verb — colour — is not present, the user can supply his own subroutines which process the MOVE, PLOT and DRAW verbs. That is possible by the monitor placing the default addresses for those routines in accessible RAM even though the routines themselves are in ROM. The user can, therefore, re-direct the interpreter to use his own supplied routines in preference, by adjusting the ROM pointers.

That technique is also available for other system-supplied features such as error-message processing, so for instance, in this case, the programmer might choose to try to recover from an otherwise fatal error alternatively, he might chose to produce a more specific message than the systems default.

In general, the documentation is very clear about the monitor and which memory locations can be manipulated to adjust its features and operation. Relevant to that, I was glad to note that the abominable Peek and Poke commands had been relegated to the scrap-heap and replaced by a much more elegant idea.

If a variable — holding a value which is a memory location — is preceded by a '?', the expression is interpreted either as a Peek or Poke depending on its context. Thus, for example, in the statement "?L = ?L + N", the "?L" at the left-hand side of the "=" is interpreted as a Poke while the other occurrence is interpreted as a Peek.

For the assembler programmer, access to the assembler is gained through Basic. Two special symbols, '[' and ']', are used to enclose assembler statements within a Basic program. When, during the run of the program, those statements are encountered, they are not executed, but assembled.

The location into which the machine code is placed is governed by the Basic variable P which is manipulated by the assembler just as a regular location counter or which can be adjusted by the Basic program. In fact, the variables available to the assembler programmer are identical to those available to the Basic programmer. Naturally, there is a Basic command available, LINK, which transfers control from Basic to the machine code, but there is a much more important use for writing assembler statements within a Basic framework.

Since the assembler blocks enclosed by the brackets assume the same status as an ordinary Basic statement, the higher-level language can control the selective assembly into machine code. That is, condition tests in Basic can determine whether or not a given section of assembler code will be included in or excluded from the assembly.

The systems developer who may wish to produce several versions of a machine-code program, each with its own local variations can therefore use Basic to formulate the individual options, and select those options required perhaps by a question-and-answer preamble at the beginning of the control program.

Using similar methods, macro statements, i.e., groups of assembler statements referenced by a name, can be built easily. The two utilities of user-defined macros and conditional assembly provide the developer with a powerful tool found normally in much larger machines.

Conclusions

• The Atom is a low-cost modular and expandable system.

• The software provided derives from a creative approach to mixing Basic and machine code, although criticism of the dialect of Basic included would be very understandable.

• Most buyers will be hobbyists, educational establishments, or perhaps engineers.

• The machine will make less impression in the immediate future on the commercial market. Two developments in the pipeline promise, however, to change that.

• The mini-floppy option should be available soon. If the company sticks to the target price of less than £200, a disc-based system with colour graphics for around £450 will certainly open new markets, making the machine very attractive to the small business user.

• The second development is a combination of several machines into a market based on the Cambridge Ring; that concept allows a large number of units to be connected as nodes on a loop, each node controlling a particular system resource — printer, hard disc, VDU etc. — resources are shared in the total system via messages which circulate around the loop.

• Current opinion is that this architecture will provide an ideal base for a distributed system such as the not-so-futuristic electronic office.

• The Atom must, therefore, be a strong contender for inclusion on the short list of a wide range of buyers.